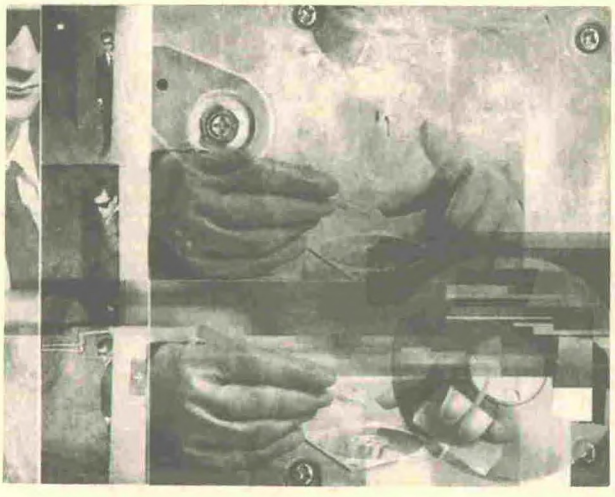
ชิตสกุณ ศุภศรี
จักรเทพ แดงอุทัย

แนะนำ Thread บน Delphi



เกร็ดความรู้เล็ก ๆ น้อย ๆ เพื่อเป็นแนวทางนำ Thread ไปใช้เป็นแนวทางในการเพิ่มประสิทธิภาพให้กับแอพพลิเคชัน

สำหรับท่านผู้เขียนโปรแกรมเกี่ยวกับ OS ก็คงรู้จักกับคำว่า Thread กันมาบ้างแล้วนะครับ แต่สำหรับท่านที่ยังไม่เคยรู้จักหรือได้ยิน ผมก็ขอแนะนำเกร็ดความรู้เล็ก ๆ น้อย ๆ เพื่อเป็นแนวทางให้คุณผู้อ่านหลาย ๆ ท่านได้นำเจ้าคำว่า Thread นี้ไปใช้เป็นแนวทางในการเพิ่มประสิทธิภาพให้กับแอพพลิเคชันต่อไป

ถ้าพูดถึงความหมายของ Thread ในความหมายเพื่อความเข้าใจแล้ว Thread จะเป็นลักษณะของโมดูลเล็ก ๆ ของ Process ที่จะนำมาแตกโมดูลเพื่อสนับสนุนลักษณะงานแบบ Multitasking ในกรณีที่เรามีหลาย ๆ โมดูลเราเรียกกรณีนี้ว่า Multithread อาจบอกได้ว่า Thread เป็นแนวทางหนึ่งในการเขียนโปรแกรมสำหรับงาน Multitasking นั่นเอง ฟังแล้วอาจจะยังไม่เข้าใจแต่ก็ขอให้อ่านกันต่อไป โดยผู้เขียนจะพยายามยกตัวอย่างและอธิบายอย่างง่าย ๆ ให้เข้าใจ

พูดถึงคำว่า Multitasking ก็คือการทำงานหลาย ๆ อย่างพร้อมกันในเวลาเดียวกันนั่นเอง แต่การทำงานแบบ Multitasking สำหรับคอมพิวเตอร์ที่แท้จริงนั้นก็ควรจะมีลักษณะที่เรียกว่า Multiprocessor ก็คือลักษณะเครื่องเดียวแต่มีหลาย CPU นั่นเอง

ทำไมถึงควรจะเป็น Multiprocessor ยกตัวอย่างเช่น เราต้องการจะยกกล่อง 2 กล่องในเวลาเดียวกัน ถ้ามีคน 2 คนช่วยกันยกกล่องคนละใบย่อมจะเป็นการทำ Multitasking ที่สมบูรณ์กว่าการที่จะใช้คน ๆ เดียวที่จะยกกล่อง 2 ใบแน่ ๆ แต่เนื่องจากลักษณะของ Personal Computer ที่นิยมใช้กันเป็นลักษณะแบบ Single Processor เปรียบเสมือนว่ามีคนที่จะทำงานเพียงคนเดียวเท่านั้นเอง ถ้าต้องการทำงานเพียงชิ้นเดียวก็คงจะไม่มีปัญหา แต่ถ้าต้องการทำงานหลาย ๆ งานในเวลาเดียวกันก็คงจะต้องเหนื่อยแน่ ๆ แต่กรณีของคอมพิวเตอร์จะทำงานในลักษณะเสมือนว่าได้ทำงานแบบ Multitasking โดยจะอาศัยความเร็วของ CPU ที่มีอยู่อย่างมหาศาลมาใช้ประโยชน์เพื่อที่จะทำงานในลักษณะนี้

OS เป็นสิ่งสำคัญ

OS (Operating System) เป็นซอฟต์แวร์ระบบปฏิบัติการของคอมพิวเตอร์ ซึ่งจะเป็นตัวกลางสำคัญ โดยจะเป็นตัวบริหารเวลาแล้วนำความเร็วของ CPU ไปแบ่งให้กับงานแต่ละงานอย่างยุติธรรม ซึ่งเป็นสิ่งที่สำคัญมากสำหรับ OS ที่สนับสนุนกับลักษณะงานประเภท Multitasking ก็จะต้องมีลักษณะการเขียนโปรแกรมที่แตกต่างจาก OS ที่ไม่ได้สนับสนุน

ปัญหาที่เกิดขึ้นจากการเขียนโปรแกรมบน DOS

DOS เป็น OS ที่มีลักษณะการทำงานแบบ Single Tasking จะทำได้ก็คงจะเป็นโปรแกรมที่มีลักษณะเป็น TSR (Terminate Stay Resident) มีลักษณะการทำงานแบบที่เรียกว่า Task Switching ไม่ใช่การทำงานแบบ Multitasking

การจะเขียนโปรแกรมแบบ Multitasking บน DOS นั้นจำเป็นต้องอาศัยเทคนิคการเขียนโปรแกรมที่ผู้เขียนโปรแกรมต้องเป็นผู้สรรค์สร้างขึ้นมาเอง ซึ่งเป็นเรื่องที่ยุ่งยากพอสมควรเลยทีเดียว และถ้าหากว่าภาษาที่ใช้เขียนโปรแกรมนั้นเป็นภาษาระดับสูงอย่างเช่น Pascal, Basic แล้ว โปรแกรมก็จะมีลักษณะไม่คล่องตัวเท่าที่ควรจะเป็น

แนวทางที่ดีกว่า

แนวทางที่ดีกว่าก็คือ OS ตัวอื่น ๆ นั่นเอง ไม่ว่าจะเป็น Windows 95, NT, UNIX หรือแม้แต่ OS/2 ล้วนแล้วแต่สนับสนุนการทำงานแบบ Multitasking ที่เป็น Thread ทั้งนั้น

การ Programming

การเขียนโปรแกรมบน OS ที่สนับสนุนการทำงานแบบ Multitasking นั้นจะแตกต่างกับการเขียนโปรแกรมในลักษณะกรณีที่เราเขียนโปรแกรมที่มีลักษณะของการวนลูปในโมดูลที่เรากำลังเขียนอยู่ ถ้าเป็นลักษณะการเขียนโปรแกรมทั่ว ๆ ไปที่เราเขียนบน DOS เราอาจจะไม่ต้องแคร์อะไร เพราะ OS ของเรานั้นเป็น DOS ซึ่งไม่ได้สนับสนุนงานประเภท Multitasking

แต่ถ้า OS ของเราเป็น Windows 32 Bit (95 หรือ NT) จะเห็นว่าการเขียนโปรแกรมธรรมดา ๆ แบบที่เขียนบน DOS ก็จะกลายเป็นปัญหาเมื่อนำไปเขียนบน Windows 95 หรือ NT จะขอยกตัวอย่างโมดูลง่าย ๆ ตัวหนึ่งตามโปรแกรม Example1

```
Procedure   Example1;
var   i : integer;
      tmp : string;
begin
      For i: = 1 to 1000  do
            begin
                  tmp :=  'Line ' + InttoStr(I);
                  Form1.Memo1.Lines.Add(tmp);
            end;
end;
```

ในกรณี Procedure Example1 นี้เราต้องการใส่ Line ลงไปในคอมโปเนนต์ Memo1 ถึง 1,000 ครั้ง ซึ่งเราจำเป็นจะต้องรอลูปในโมดูลทำการ Add จนถึง 1,000 ครั้ง โดยที่เราไม่สามารถจะทำงานอื่น ๆ ได้เลยบนแอพพลิเคชันของเรา หรือแม้แต่จะ Minimize Application ของเรา เพื่อเรียกทำงานจากแอพพลิเคชันอื่น ๆ ซึ่งก็จะกลายเป็นลักษณะ Single Tasking ที่อยู่บน DOS ไป แต่ถ้า OS ของเราเป็น Windows 95 หรือ Windows NT ซึ่งเป็นระบบ OS ที่สนับสนุนการทำงานแบบ Multitasking (Preemtive Multitasking) เราก็จะแก้ปัญหาเบื้องต้นนี้ได้

แก้ปัญหาเฉพาะหน้าด้วย Application.ProcessMessage.

```
procedure  Example2;
begin
      for  i := 1  to 1000  do
           begin
            tmp :=  'Line ' + InttoStr(I);
            Form1.Memo1.Lines.Add(tmp);
            Application.ProcessMessages;
           end;
end;
```

ในลูปจะมีคำสั่ง Application.ProcessMessages เพิ่มขึ้นมา ก่อนจะพูดถึงหน้าที่ของคำสั่งนี้จะขอพูดถึงเรื่องของ Message ก่อน

การทำงานของ Window ไม่ว่าจะทำงานใดก็ตาม Window จะมองเห็นงานนั้นในรูปของ Message โดย Window จะนำ Message เหล่านี้ไปใส่ไว้ในคิวเพื่อทำการเรียงลำดับการทำงานให้ถูกต้อง คำสั่ง Application.ProcessMessages ก็จะเป็นการเก็ตค่าของ Message เหล่านั้นออกจากคิวแล้วนำไปประมวลผลนั่นเอง ซึ่งเมื่อเราแทรกคำสั่ง Application.ProcessMessages นี้ลงไปแล้วทำการรันโปรแกรม เมื่อ CPU ทำงานจนมาถึงบรรทัดนี้ Application.ProcessMessages จะเป็นตัวบอกให้ Window นั้นวิ่งไปตรวจสอบว่ามีเหตุการณ์ (Event) ที่คิดว่ามี Message เกิดขึ้นหรือเปล่า ถ้ามีก็จะทำการตอบสนอง Message นั้นก่อน (ไม่ว่าจะเป็นเหตุการณ์ที่เราโปรแกรมขึ้นมา หรือจะเป็นเหตุการณ์ที่เกิดขึ้นจากตัวของ OS เองก็ตาม) และจะทำจนกระทั่งได้รับการ Return ค่ากลับของ Message ที่ CPU นั้นได้แวะไปทำมา CPU ถึงจะกลับไปทำงานในลูปต่อ

จะเห็นได้ว่าเมื่อเราทำการแทรกคำสั่ง Application.Process Messages ไปแล้ว เราก็สามารถจะทำงานอย่างอื่นได้ในขณะที่ลูปนี้วิ่งอยู่ได้อีกขั้นหนึ่ง แต่วิธีการนี้จะทำให้เกิด Delay ขึ้นในลูป ส่วน Delay ที่เกิดขึ้นนี้จะยาวนานเท่าใดนั้นก็ขึ้นอยู่กับว่างานที่เกิดจากเหตุการณ์ที่เกิดขึ้นนั้นจะส่งค่า Return กลับเมื่อใด จากเหตุผลนี้แสดงให้เห็นว่าวิธีการนี้เป็นวิธีการที่ไม่ดีเท่าที่ควร เพราะจะทำให้เกิด Delay ที่ไม่แน่นอน และยังเป็นการยากในการควบคุมการทำงานของ Multitasking ที่มีโมดูลหลาย ๆ ตัว โดยเฉพาะถ้า OS ของเรามีการทำงานแบบ Preemtive Multitasking วิธีการนี้จะถือว่าเป็นการแก้ปัญหาเฉพาะหน้าเท่านั้น เพราะวิธีการนี้เป็นวิธีการแก้ปัญหาแบบที่ใช้ใน Windows 3.X 16 bit (Co-operative Multitasking) ถ้า OS ของเราสนับสนุนการทำงานแบบ Preemtive Multitasking (Windows 95 และ NT) เพราะฉะนั้นเราจึงควรหาเทคนิคอื่นมาใช้แทน

ใช้ Timer ในการแก้ปัญหา

อย่างที่ได้กล่าวไปแล้วว่าวิธีนี้ไม่ใช่วิธีการที่ดีสักเท่าไรนัก แต่ยังมีเทคนิคอีกอย่างที่นิยมใช้กัน นั่นก็คือการใช้ Timer Component ที่อยู่ใน Palatte System แทนการใช้ลูป ดังในโมดูลที่ยกตัวอย่างก็จะกลายเป็น

```
var I:integer=0;  { Global }

Procedure  TForm1.Example3Timer(Sender : Tobject);
var tmp:string;
begin
      if I=1000 then Timer1.Enable:=false
      else   inc(I);
      tmp :=  'Line ' + InttoStr(I);
      Memo1.Lines.Add(tmp);
end;
```

จากตัวโปรแกรม Procedure TForm1.Example3Timer จะเห็นว่าเป็นการใช้ Timer แทนลูปนั่นเอง เมื่อรันโปรแกรมก็จะสามารถแก้ปัญหาที่เกิดขึ้นได้เช่นเดียวกัน วิธีนี้ถึงแม้ว่าเราสามารถที่จะเซตค่าของ Interval ใน Property เพื่อปรับเปลี่ยน Delay ได้ก็ตาม ถ้าจุดประสงค์ของโปรแกรมไม่ได้มีความเดือดร้อนกับ Delay ที่เกิดขึ้นหรือเป็นโปรแกรมที่ต้องการ Delay อยู่แล้ว วิธีนี้ก็ดูจะเป็นวิธีที่น่าสนใจอยู่เหมือนกัน แต่ถ้าเป็นโปรแกรมที่ต้องการให้ทำงานเร็วที่สุดเท่าที่จะทำได้ (มี Delay น้อยที่สุด) วิธีนี้ก็ยังคงมีปัญหาอยู่เพราะจะเกิด Delay ที่ไม่เหมาะสม โดยเฉพาะถ้าการทำงานในลูปนั้นใช้เวลาในการทำงานในลูป 1 ครั้งมากกว่า Interval ที่เซตไว้ ก็จะเกิด Error ขึ้น ยกตัวอย่างโปรแกรม Example4

```
procedure Example4(Line_num:integer);
var   i,j:integer;
      Step:double;
begin
      Step:=Form1.Width / 256;
      for i:=0 to Form1.Height do
        for j:=0 to Form1.Width do
        Form1.Canvas.Pixels[i,j]:=RGB(Round(i*j/
        Step),Round((i*j)/Step),Round(i*j));
end;
```

แล้วเราก็แทน Loop for นอกด้วย Timer ก็จะกลายเป็นดังเช่นโปรแกรม Procedure TForm1.Example5Timer

```
var I:integer=0;  { Global }

Procedure  TForm1.Example5Timer(Sender : TObject);
var j:integer;
begin
   if I=Form1.Height  then  Timer1.Enable:=false
      else   inc(I);
   for j:=0 to Form1.Width do
      Form1.Canvas.Pixels[i,j]:=RGB(Round(i*j/
      Step),Round((i*j)/Step),Round(i*j));
end;
```

ลูปนี้เป็นการแสดงสีหลาย ๆ สีของ Function RGB ลงบน Form1 ทีละ 1 พิกเซล จึงทำให้ต้องใช้เวลานาน แต่ที่ใช้ Timer แทนลูปนั้นเป็นการแทนการลงสีครั้งละ 1 เส้นด้วยลูป 1 ลูป ถ้าจุดประสงค์ของโปรแกรมนี้ต้องการให้เกิด Delay ที่น้อยที่สุด ดังนั้นเราก็จะพยายามเซตค่าของ Interval ให้น้อยที่สุดเท่าที่จะทำได้ แต่ถ้าเราเซตค่าไว้น้อยกว่าที่ควรจะเป็น คือน้อยกว่าเวลาที่คำสั่งเหล่านี้จะทำงานหมดทุกคำสั่ง CPU ก็จะทำงานไม่ทันกับ Timer ที่ Toggle อยู่ตลอดเวลา และสิ่งที่จะตามมาก็คือการเกิด Error ของโปรแกรมนั่นเอง

แก้ด้วย Thread ดีที่สุด

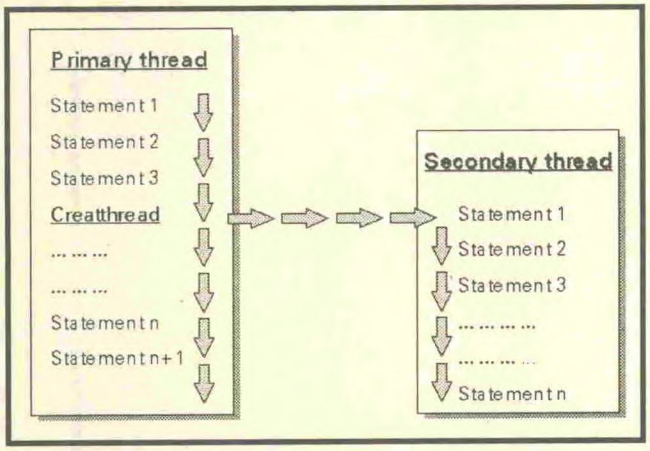
ทางแก้ไขที่ดีที่สุดนั้นก็คือ "Thread" นั่นเอง เพราะเมื่อเราใช้ Thread นั้น OS ของเราก็จะแตก Process หรือโมดูลของเราออกบริหารของเราให้เกิด Delay (น้อยที่สุดเท่าที่จะทำได้) กับสภาพการทำงานของโปรแกรม การสร้าง Thread ขึ้นมานั้นไม่ใช่เรื่องยาก ดังตัวอย่าง

```
procedure TForm1.Button1Click(Sender: TObject);
var   DW : DWORD;
begin
      CreateThread(nil, //   Thread Atttribute
        0,              //   Stack Size
        @Example1,      //   Start  Address
        nil,            //   Thread  Parameter
        0,              //   Flag
        DW);            //   Thread ID
end;
```

คำสั่ง CreateThread เป็นคำสั่งที่เราบอก OS ว่าให้ทำการเปิด Thread สำหรับ procedure Example1 ขึ้นมา ผลที่ได้ก็คือเราสามารถจะทำงานอย่างอื่นได้อีกในขณะที่ Procedure Example1 กำลังทำงานอยู่

โดยปกติแล้วโปรแกรมบน Windows ถ้าไม่มีการ Createthread ก็จะถือว่าเป็น Thread อยู่แล้ว เราจะเรียกว่า Main Thread, Primary Thread หรือ Default Thread ในกรณีที่เราเขียนโปรแกรมปกตินั่นคือการที่เราใช้ Main Thread ในการจัดการ ซึ่งก็จะเกิดปัญหาตามที่กล่าวมาข้างต้น แต่หากเรา Createthread Thread ที่ถูกสร้างขึ้นมาเราจะเรียกว่า Secondary Thread ซึ่งก็สามารถแก้ปัญหาในส่วนนี้ได้ เพราะ OS ของเราจะทำงานทั้ง Main Thread และ Secondary Thread **เสมือนว่า Thread ทั้งสองทำงานพร้อมกัน** และเรายังสามารถที่จะสร้าง Secondary Thread ออกมาอีกหลาย ๆ ตัวได้ เพื่อให้แอพพลิเคชันของเรามีลักษณะการทำงานแบบ MultiThread เราก็สามารถทำได้

ในการ Create Thread ขึ้นมาหลาย ๆ Thread นั้น ทรัพยากรบนเครื่องคอมพิวเตอร์จะถูกแชร์ตามแต่ละ Thread ที่เกิดขึ้นมา จะเห็นว่า Procedure Example1 นั้นมีตัวแปร Local 2 ตัว โดยปกติแล้วตัวแปรเหล่านี้จะถูกเก็บไว้ใน Stack แต่ในกรณีที่ถูกเปิดเป็น Thread Stack และ Memory ทั้งหมดก็จะถูกแบ่งออกไปแชร์ให้ Thread แต่ละตัวสามารถใช้ทรัพยากรที่เป็นส่วนของตัวเองได้ โดยไม่ไปใช้ทรัพยากรทับกับของ Thread อื่น ๆ



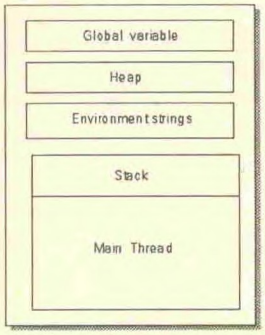
รูปที่ 1 แสดงลักษณะการเปิด Thread

เมื่อมีการเปิด Thread แล้ว ก็จะมีลักษณะเป็นดังรูปที่ 3

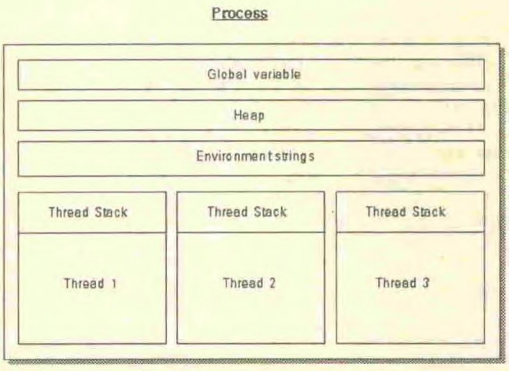
เพื่อให้แต่ละ Thread ที่ถูกสร้างขึ้นมานั้นมีตัวแปรแต่ละตัวเป็นของตัวเอง ใน Delphi สามารถที่จะกำหนดตัวแปรสำหรับ Therad แต่ละ Thread เลยได้โดยใช้ TLS (Thread Local Stroage) โดยเราจะประกาศตัวแปรชนิดนี้เป็นลักษณะของ Global variable โดยแต่ละ Thread จะทำการก๊อบปี้ตัวแปรในตัวอย่างมาเป็นตัวแปร Local ของตัวเอง โดยใช้คำสงวน ThreadVar

```
ThreadVar
      i : integer ;
      image : TImage;
```

ในทางทฤษฎีเรารู้ว่าในการใช้ Thread หลาย ๆ ตัวนั้น CPU จะสามารถทำงานหลาย ๆ อย่างได้ในเวลาเดียวกัน แต่ในทางปฏิบัตินั้นจริง ๆ แล้ว CPU สามารถทำงานได้เพียง 1 งานในขณะเวลาหนึ่งเท่านั้น โดย CPU นั้นจะทำการ Swap ตัวของมันเองเพื่อทำงานในแต่ละ Thread ที่ถูกเปิดขึ้นมา OS จะทำการบริหารเวลาที่จะทำงานของแต่ละ Thread โดยที่ตัว OS จะสามารถทำการ Preempt (Interrupt) Thread ที่กำลัง Active อยู่เพื่อที่จะแทรกการทำงานที่สำคัญกว่าหรือมีอภิสิทธิ์ (priority) ที่สูงกว่าก่อนได้ ในขณะเดียวกัน Thread ที่ถูก Preempt นั้นก็ยังคงรอ (Suspended) เพื่อที่จะกลับมาใช้งาน CPU ต่อไป เพราะฉะนั้นเราจะเห็นว่าเมื่อเราเปิด Thread ขึ้นมาหลาย ๆ Thread การทำงานของแต่ละ Thread ก็จะช้าลงอย่างเห็นได้ชัด เพราะ CPU จะต้องทำการ Swap ตัวเองเพื่อที่ทำงานที่แต่ละ Thread ส่งมาให้ได้อย่างเหมาะสมที่สุด ทั้งนี้ MultiThread จะทำงานได้ดีมากน้อยเพียงไรก็จะขึ้นอยู่กับสภาพแวดล้อมและทรัพยากรของ CPU เป็นสำคัญ



รูปที่ 2 ลักษณะ Process ปกติ



รูปที่ 3 ลักษณะ Process ที่เปิด Thread แล้ว

เมื่อ OS มีคุณสมบัติของ Thread ก็ทำให้ผู้เขียนโปรแกรมสามารถเขียนโปรแกรมในการทำงานหลาย ๆ อย่างพร้อมกันได้ง่ายขึ้น และเมื่อเราได้รู้ว่าตัว OS ต้องทำการ Swap ตัวเองเพื่อไปทำงานหลาย ๆ งาน ตัวโปรแกรมที่ทำงานแบบ Thread ก็ควรจะสนับสนุนการทำงานแบบ MultiThread ด้วย คือควรจะมีลักษณะในตัวของโปรแกรมเอง และสิ่งที่กล่าวมาข้างต้นก็จะสามารถทำได้อย่างสะดวกสบายด้วยการเขียนโปรแกรมลักษณะของ OOP (Object Oriented Programming) ซึ่งตัว Delphi เองก็สนับสนุนลักษณะการทำงานแบบนี้อยู่แล้ว

Thread บน Delphi นั้นจะต้องจะเป็น Delphi 2 หรือ 3 เพราะ Delphi 1 นั้นเป็นซอฟต์แวร์ 16 บิตที่รันบน Windows 3.x ซึ่งมีลักษณะการทำงานแบบ Non Preemtive Multitasking หรือ Co-operative Multitasking ซึ่งไม่สนับสนุนการทำงานของ MultiThread

โปรแกรมตัวอย่าง

ผู้เขียนได้เขียนตัวอย่างโปรแกรม Thread แบบง่าย ๆ โดยใช้คำสั่ง CreateThread เพื่อให้ผู้อ่านได้เห็นเส้นทางเดินของ Thread จริง ๆ ว่าเมื่อเราเปิด Thread หลาย ๆ ตัว โปรแกรมจะมีลักษณะการทำงานเป็นอย่างไร โดยจะนำตัวอย่างจาก Procedure Example4 มาดัดแปลงให้เห็นชัดเจนมากขึ้น

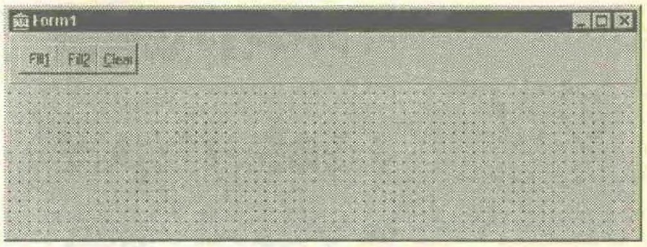
```
unit Example6;
{  Chitsakun  Suphasri &  Juktep  Dangautai  , Computer  Science , KMITNB  }

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Buttons, ExtCtrls;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Fill1: TSpeedButton;
    Fill2: TSpeedButton;
    Clear: TSpeedButton;
    procedure ClearClick(Sender: TObject);
    procedure Fill1Click(Sender: TObject);
    procedure Fill2Click(Sender: TObject);
  private
    { Private declarations }
  public
     FillCase:byte;
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.DFM}
procedure TForm1.ClearClick(Sender: TObject);
begin
     with Canvas do
        begin
           Pen.Color:=clBtnFace;
           Brush.Color:=clBtnFace;
           Rectangle(0,0,Width,Height);
        end;
end;

procedure Fill;
var i,j:integer;
    Step:double;
    FC:byte;
begin
     with Form1 do
          begin
               Step:=Width/256;
               FC:=FillCase;
               for i:=0 to Width do
                    for j:=0 to Height do
                         case FC of
                              1: Canvas.Pixels[i,j]:=RGB(Round(i/Step),Round(j/Step),0);
                              2: Canvas.Pixels[i,j]:=RGB(Round(i/Step),0,Round(j/Step));
                         end;
          end;
end;

procedure TForm1.Fill1Click(Sender: TObject);
var THID:DWORD;
begin
     FillCase:=1;
     CreateThread(Nil,0,@Fill,Nil,0,THID);
end;

procedure TForm1.Fill2Click(Sender: TObject);
var THID:DWORD;
begin
     FillCase:=2;
     CreateThread(Nil,0,@Fill,Nil,0,THID);
end;
end.
```

โดยใน Unit Example6 จะเป็นการลงสีบนพื้น Window จากฟังก์ชัน RGB ทีละ 1 พิกเซล ซึ่งเมื่อในขณะที่เรากดปุ่มลงสี Window แล้วก็จะเห็นลักษณะของการทำงานของแต่ละ Thread ที่ถูกเปิดขึ้นมา และยังสามารถที่จะเปิด Thread ขึ้นมาให้ทำงานอีกตัวในขณะที่ Thread แรกกำลังทำงานอยู่ ตัวแปร FillCase เป็นตัวแปรที่ใช้สำหรับการผ่านค่าใน Procedure Fill เพื่อเลือกลักษณะของสีที่จะระบายพื้น Window ส่วนใน Procedure TForm1.Fill1Click นั้นเป็นจะเป็นการเปิด Thread เพื่อระบายสีแบบที่หนึ่ง โดยจะเลือกลงสีโดยการผ่านค่าให้ตัวแปร FillCase แล้วก็จะทำการเปิด Thread ให้ทำงานใน Procedure Fill และใน Procedure TForm1.Fill2Click ก็มีลักษณะการทำงานเช่นเดียวกันกับ Procedure TForm1.Fill1Click แต่จะเลือกผ่านค่าให้ FillCase เป็น 2 เพื่อจะเปิด Thread ให้ Procedure Fill ทำการระบายสีในรูปแบบที่สอง ส่วน Procedure TForm1.ClearClick ก็จะเป็นการคืนสีเดิมให้กับ Window เท่านั้นเอง

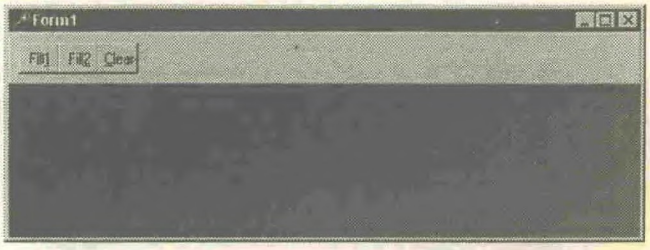
เมื่อดูจากตัวโปรแกรม Unit Example6 จะเห็นว่า Procedure TForm1.Fill1Click และ Procedure TForm1.Fill2Click จะเป็นการทำงานในลักษณะของการเปิด Secondary Thread แต่สำหรับ Procedure TForm1.ClearClick จะเป็นการทำงานบน Main Thread เพราะการทำงานใน Procedure TForm1.ClearClick มีลักษณะการทำงานที่สั้น สามารถที่จะ Interact กับผู้ใช้ได้ทันที แต่ต่างจาก Procedure TForm1.Fill1Click และ ProcedureTForm1.Fill2Click ซึ่งมี ลูปในการทำงานจำเป็นต้องให้ USER รอการทำงานของโปรแกรม การเขียนโปรแกรมในส่วนนี้จึงต้องทำการเปิดเป็น Secondary Thread ขึ้นมาเพื่อให้ USER สามารถที่จะทำงานอื่น ๆ ได้ในขณะที่โปรแกรมกำลังทำงานอยู่ในส่วนนี้

ถ้าเรารันโปรแกรมนี้ในขณะที่เรากำลังรันโปรแกรมอื่นอยู่ด้วย โปรแกรมนี้ก็จะทำงานช้าลงไป แต่สำหรับโปรแกรมนี้การทำงานของมันยังไม่ใช่ลักษณะที่สมบูรณ์นัก เพราะยังขาดปัจจัยสำคัญอยู่ 2 อย่าง อย่างแรกก็คือการโปรแกรมที่เป็น MultiThread ควรจะเป็นการเขียนโปรแกรมประเภท OOP ซึ่งใน Unit Example6 ยังไม่ใช่ OOP แต่เป็นการใช้ประโยชน์จากคำสั่ง CreateThread เท่านั้น และอย่างที่สองที่สำคัญมากก็คือการควบคุมแต่ละ Thread ที่ทำงานอยู่ไม่ให้ก้าวก่ายการทำงานของ Thread อื่น ๆ ที่กำลังทำงานอยู่หรือที่เราเรียกว่าการทำ Synchronization ซึ่งทั้ง 2 เรื่องเป็นเรื่องใหญ่พอสมควรในการจะพูดให้เข้าใจ ถ้ามีโอกาสในคราวหน้าก็จะขอนำทั้ง 2 เรื่องนี้มากล่าวถึง

Thread เป็นสิ่งที่มีความสำคัญสำหรับผู้เขียนโปรแกรมใช้งานบน Window เลยทีเดียวก็ว่าได้ เพราะไม่เพียงแต่ช่วยให้เราสามารถทำงานหลาย ๆ อย่างได้พร้อม ๆ กัน แต่ยังเป็นการเขียนโปรแกรมเพื่อรองรับความสามารถของ CPU ที่มีความเร็วสูงขึ้นเรื่อย ๆ โดยนำความเร็วของ CPU ที่มีอยู่อย่างมหาศาลมาใช้ให้เกิดความประโยชน์คุ้มค่าที่สุด



รูปที่ 4 แสดง Form ก่อนรันโปรแกรม



รูปที่ 5 แสดง Form หลังรันโปรแกรม

เอกสารอ้างอิง

1. Tom Swan, Jeff Cogswell. **Delphi 32-Bit Programming SECRETS.** ODG Books Worldwide, Inc. An International Data Group Company.

2. Dan Osier, Steve Grobman and Steve Batson. **Teach Yourself Delphi 2 in 21 days** First edition 201 West 103rd Street Indianapolis, Indiana 46290: Sams net

3. Ray Lischner. **Secrets of Delphi 2.** WAIT GROUP PRESS, A Division of Sams Publishing, Corte Madera, CA.